

```
SSSSSSSSSSSSS  MMM      MMM      GGGGGGGGGGGG  RRRRRRRRRRRR  TTTTTTTTTTTTTT  LLL
SSSSSSSSSSSSS  MMM      MMM      GGGGGGGGGGGG  RRRRRRRRRRRR  TTTTTTTTTTTTTT  LLL
SSSSSSSSSSSSS  MMM      MMM      GGGGGGGGGGGG  RRRRRRRRRRRR  TTTTTTTTTTTTTT  LLL
SSS            MMMMMM  MMMMMM  GGG            RRR      RRR      TTT      LLL
SSS            MMMMMM  MMMMMM  GGG            RRR      RRR      TTT      LLL
SSS            MMMMMM  MMMMMM  GGG            RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG            RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG            RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG            RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG            RRR      RRR      TTT      LLL
SSSSSSSSSSS    MMM      MMM      GGG            RRRRRRRRRRRR  TTT      LLL
SSSSSSSSSSS    MMM      MMM      GGG            RRRRRRRRRRRR  TTT      LLL
SSSSSSSSSSS    MMM      MMM      GGG            RRRRRRRRRRRR  TTT      LLL
SSS            MMM      MMM      GGG      GGGGGGGGGG  RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG      GGGGGGGGGG  RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG      GGGGGGGGGG  RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG      GGG      GGG  RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG      GGG      GGG  RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG      GGG      GGG  RRR      RRR      TTT      LLL
SSS            MMM      MMM      GGG      GGG      GGG  RRR      RRR      TTT      LLL
SSSSSSSSSSSSS  MMM      MMM      GGGGGGGGGG  RRR      RRR      TTT      LLLLLLLLLLLLLLLL
SSSSSSSSSSSSS  MMM      MMM      GGGGGGGGGG  RRR      RRR      TTT      LLLLLLLLLLLLLLLL
SSSSSSSSSSSSS  MMM      MMM      GGGGGGGGGG  RRR      RRR      TTT      LLLLLLLLLLLLLLLL
```



```
SSSSSSSS MM MM GGGGGGGG MM MM AAAAAA CCCCCCCC RRRRRRRR 000000 SSSSSSSS
SSSSSSSS MM MM GGGGGGGG MM MM AAAAAA CCCCCCCC RRRRRRRR 000000 SSSSSSSS
SS SS MMMM MMMM GG MM MM AA AA CC CC RR RR 00 00 SS
SS SS MMMM MMMM GG MM MM AA AA CC CC RR RR 00 00 SS
SS SS MM MM GG MM MM AA AA CC CC RR RR 00 00 SS
SSSSSS SS MM MM GG GGGGGG MM MM AAAAAAAA CC CC RRRRRRRR 00 00 SSSSSS
SSSSSS SS MM MM GG GGGGGG MM MM AAAAAAAA CC CC RRRRRRRR 00 00 SSSSSS
SS SS MM MM GG GG GG MM MM AAAAAAAA CC CC RR RR 00 00 SS
SS SS MM MM GG GG GG MM MM AAAAAAAA CC CC RR RR 00 00 SS
SS SS MM MM GG GG GG MM MM AAAAAAAA CC CC RR RR 00 00 SS
SSSSSSSS MM MM GGGGGG MM MM AA AA CCCCCCCC RR RR 000000 SSSSSSSS
SSSSSSSS MM MM GGGGGG MM MM AA AA CCCCCCCC RR RR 000000 SSSSSSSS
.....
RRRRRRRR EEEEEEEEE EEEEEEEEE QQQQQQ
RRRRRRRR EEEEEEEEE EEEEEEEEE QQQQQQ
RR RR EE EE QQ QQ
RR RR EE EE QQ QQ
RR RR EE EE QQ QQ
RRRRRRRR EEEEEEEEE EEEEEEEEE QQ QQ
RRRRRRRR EEEEEEEEE EEEEEEEEE QQ QQ
RR RR EE EE QQ QQ
RR RR EE EE QQ QQ
RR RR EE EE QQ QQ
RR RR EE EE QQ QQ
RR RR EEEEEEEEE EEEEEEEEE QQQQ QQ
RR RR EEEEEEEEE EEEEEEEEE QQQQ QQ
```



```

*****
*
*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
*  ALL RIGHTS RESERVED.
*
*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
*  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
*  TRANSFERRED.
*
*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
*  CORPORATION.
*
*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*
*****

```

! T
XIF
XTH
XFI

++

FACILITY: Screen Management

ABSTRACT:

This file contains macros used by screen management routines.

MODIFIED BY:

- 1-001 - Original. PLL 15-Mar-1983
- 1-002 - Add \$SMG\$VALIDATE_ARGCOUNT. PLL 24-Mar-1983
- 1-003 - Add \$SMG\$VALIDATE_ROW_COL. PLL 7-Apr-1983
- 1-004 - Add lots more. PCL 14-Apr-1983
- 1-005 - Add \$SMG\$FIND_PRINT_LENGTH. PLL 21-Apr-1983
- 1-006 - Fix \$SMG_GET_PBCB so that it can work in any module.
STAN 3-May-1983
- 1-007 - Fix \$SMG\$FIND_PRINT_LENGTH. PLL 12-May-1983
- 1-008 - Delete \$SMG\$CHECK_FOR_WRAP since it's not used.
PLL 18-May-1983
- 1-009 - Change to make display id's really be DCB addresses.
Deleted \$SMG\$GET_NEXT_DID -- no longer needed.
Changed \$SMG\$GET_DCB.
RKR 20-May-1983.
- 1-010 - Add input macros. (Formerly in SMGLIB.REQ.) PLL 21-Jun-1983
- 1-011 - Fix SET_REND_CODE. STAN 23-Jun-1983.
- 1-012 - Add \$SMG\$GET_TERM_DATA. STAN 15-Jan-1984.

SSMG\$VALIDATE_ARGCOUNT

Macro used to check that a SMGS procedure was called with the correct number of arguments. If the test fails, the procedure returns with the failure status SMGS_WRONUMARG.

Format:

SSMG\$VALIDATE_ARGCOUNT (lo, hi);

lo = Lowest number of arguments which are valid (0-255)
hi = Highest number of arguments which are valid (0-255)

MACRO

```
SSMG$VALIDATE_ARGCOUNT (lo, hi) =  
  BEGIN  
  BUILD IN  
  ACTUALCOUNT;  
  EXTERNAL LITERAL  
  SMGS_WRONUMARG;  
  
  %IF lo NEQ hi  
  %THEN  
    %IF lo NEQ 0  
    %THEN  
      LOCAL  
      DIFF: BYTE;  
      DIFF = ACTUALCOUNT () - lo;  
      IF .DIFF GTRU (hi - lo)  
      THEN  
        RETURN SMGS_WRONUMARG;  
    %ELSE  
      IF ACTUALCOUNT () GTRU hi  
      THEN  
        RETURN SMGS_WRONUMARG;  
    %FI  
  
  %ELSE  
    IF ACTUALCOUNT () NEQU lo  
    THEN  
      RETURN SMGS_WRONUMARG;  
  %FI  
  
  END %;
```

SSMG\$VALIDATE_ROW_COL

This macro checks to make sure the specified row and column are within the virtual display. SMGS_INVROW or SMGS_INVCOL is returned if they are not.

```
MACRO SSMG$VALIDATE_ROW_COL (ROW, COL) =  
  BEGIN
```

```
EXTERNAL LITERAL SMGS_INVROW;
EXTERNAL LITERAL SMGS_INVCOL;
IF ROW LEQ 0 OR
  ROW GTR .DCB [DCB_W_NO_ROWS]
THEN
  RETURN (SMGS_INVROW);

IF COL LEQ 0 OR
  COL GTR .DCB [DCB_W_NO_COLS]
THEN
  RETURN (SMGS_INVCOL);

END; %;
```

↑ \$SMGSGET_DCB

Macro \$SMGSGET_DCB validates the supplied virtual display id (DID) and computes the starting address of the corresponding Display Control Block (DCB). If DID is invalid, SMGS_INVDIS_ID is returned to caller of routine that invokes this macro.

MACRO

```
$SMGSGET_DCB ( DID, DCB_ADDR) =
BEGIN
  BIND LOC_DID = DID : REF BLOCK [,BYTE];
  EXTERNAL LITERAL SMGS_INVDIS_ID;

  IF .LOC_DID [DCB_L_DID] NEQ .DID
  THEN
    RETURN (SMGS_INVDIS_ID);      ! Not pointing to one of our
                                ! control blocks

  IF .LOC_DID [DCB_B_STRUCT_TYPE] NEQ DCB_K_STRUCT_TYPE
  THEN
    RETURN (SMGS_INVDIS_ID);      ! Not pointing to a DCB

  DCB_ADDR = .DID;                ! Assume ok
END; %;
```

↑ \$SMGSGET_PBCB

Macro \$SMGSGET_PBCB validates the supplied pasteboard id (PID) and computes the starting address of the corresponding Pasteboard Control Block (PBCB). If PID is invalid, SMGS_INVPAS_ID is returned to caller of routine that invokes this macro.

MACRO

```
$SMGSGET_PBCB ( PID, PBCB_ADDR) =
BEGIN
  EXTERNAL LITERAL SMGS_INVPAS_ID;

  %IF NOT %DECLARED ( PBD_L_COUNT )
  %THEN
```



```

EXTERNAL PBD_L_COUNT,
          PBD_A_PBCB   : VECTOR,
          PBD_V_PB_AVAIL : BITVECTOR;

%FI
IF .PID LSS 0          OR      : Too small
.PID GTR .PBD_L_COUNT OR      : Too big
NOT .PBD_V_PB_AVAIL [.PID]    : Not allocated
THEN
  RETURN (SMGS_INVPAS_ID);

PBCB_ADDR = .PBD_A_PBCB [ .PID];
END; %;

```

* \$SMGSGET_NEXT_PID

Macro \$SMGSGET_NEXT_PID attempts to allocate the next available pasteboard id (PID). If no more can be allocated, it returns SMGS_TOOMANPAS to the caller of the routine that invoked this macro. If a new one can be allocated, it is returned as PID and the corresponding bit in PBD_V_PB_AVAIL is set to one to indicate that this number is in use.

MACRO

```

$SMGSGET_NEXT_PID (PID)=
BEGIN
BUILTIN
FFC;
EXTERNAL LITERAL SMGS_TOOMANPAS;

IF .PBD_L_COUNT + 1 GTR PBD_K_MAX_PB
THEN
  RETURN (SMGS_TOOMANPAS);

```

If we've done our bookkeeping correctly, the FFC should find a zero within PBD_K_MAX_PB bits.

```

FFC ( ZERO,          : Starting bit position
      PBD_K_MAX_PB_BY_REF, : Number of bits to search
      PBD_V_PB_AVAIL,   : Base of search
      PID);            : Position of 1st zero bit found

```

```

PBD_V_PB_AVAIL [ .PID] = 1; ! Mark as allocated
END; %;

```

* \$SMGSGET_TERM_DATA

Gets terminal data from a terminal table into the PBCB. Assumes you have a symbol named "PBCB".

MACRO

```
SSMSG$GET_TERM_DATA(CAPABILITY,ARG1,ARG2) =
```

```
BEGIN
```

```
LOCAL
```

```
    INPUT_ARGS      : VECTOR[3],  
    STATUS;
```

```
EXTERNAL ROUTINE
```

```
    SMG$GET_TERM_DATA;
```

```
IF .PBCB[PBCB_L_TERMTABLE] EQL 0
```

```
THEN PBCB[PBCB_L_CAP_LENGTH]=0
```

```
ELSE BEGIN
```

```
    INPUT_ARGS[0]=%LENGTH-1;
```

```
    %IF NOT %NULL(ARG1)
```

```
        %THEN INPUT_ARGS[1]=ARG1
```

```
    %FI;
```

```
    %IF NOT %NULL(ARG2)
```

```
        %THEN INPUT_ARGS[2]=ARG2
```

```
    %FI;
```

```
    STATUS=SMG$GET_TERM_DATA(PBCB[PBCB_L_TERMTABLE],  
                             %REF( %NAME(SMG$K_CAPABILITY) ),  
                             PBCB[PBCB_L_LONGEST_SEQUENCE],  
                             PBCB[PBCB_L_CAP_LENGTH],  
                             PBCB[PBCB_A_CAP_BUFFER],  
                             INPUT_ARGS);
```

```
IF NOT .STATUS THEN RETURN .STATUS  
END
```

```
END
```

```
%;
```


Macros for manipulation queue entries

SSMG\$INSET_AT_HEAD

Macro SSMG\$INSERT_AT_HEAD inserts the specified queue entry in the queue at a position now occupied by the entry pointed to by the forward pointer part of the queue header.

```
MACRO
  SSMG$INSERT_AT_HEAD ( Q_ENTRY, Q_HEAD ) =
  BEGIN
    BUILTIN INSQUE;
    INSQUE ( Q_ENTRY, Q_HEAD );
  END; %;
```

SSMG\$INSET_AT_TAIL

Macro SSMG\$INSERT_AT_TAIL inserts the specified queue entry in the queue at a position now occupied by the entry pointed to by the backward pointer part of the queue header.

```
MACRO
  SSMG$INSERT_AT_TAIL ( Q_ENTRY, Q_HEAD ) =
  BEGIN
    BUILTIN INSQUE;
    INSQUE ( Q_ENTRY, .(Q_HEAD +4));
  END; %;
```

SSMG\$REMOVE_FROM_QUEUE

Macro SSMG\$REMOVE_FROM_QUEUE removes the specified entry from the queue.

```
MACRO
  SSMG$REMOVE_FROM_QUEUE ( Q_ENTRY ) =
  BEGIN
    BUILTIN REMQUE;
    LOCAL
      FOO;      ! Item to be thrown away
    REMQUE ( Q_ENTRY, FOO );
  END; %;
```

SSMG\$Linear

Macro SSMG\$Linear linearizes a two dimensional subscript formed by a 1-based row number and a 1-based column number, into a single 0-based subscript.

```
MACRO
  SSMG$LINEAR (ROW_NUMBER, COLUMN_NUMBER) =
  (ROW_NUMBER-1)*DCB [DCB_W_NO_COLS] + COLUMN_NUMBER -1 %;
```

!+
\$SMG\$Set_rend_code

This macro sets the rendition code by checking for optional rendition set
and rendition_complement arguments, and using the default from the DCB.

```
MACRO $SMG$SET_REND_CODE (SET_ARG_NO, COMP_ARG_NO) =
  BEGIN
    BUILTIN
      NULLPARAMETER;
      REND_CODE = .DCB [DCB_B_DEF_VIDEO_ATTR];
      IF NOT NULLPARAMETER (SET_ARG_NO)
      THEN
        REND_CODE = .REND_CODE OR ..RENDITION_SET;
      IF NOT NULLPARAMETER (COMP_ARG_NO)
      THEN
        REND_CODE = .REND_CODE XOR ..RENDITION_COMPLEMENT;
  END; %;
```

!+
\$SMG\$Blank_fill_DCB

This macro puts blanks into the text buffer, the default attribute byte
into the attribute buffer, and the default character set byte into the
character set buffer.

```
MACRO $SMG$BLANK_FILL_DCB (NUM, SRC) =
  BEGIN
    LOCAL
      TEXT_BUF : REF VECTOR [,BYTE],
      ATTR_BUF : REF VECTOR [,BYTE],
      CHAR_BUF : REF VECTOR [,BYTE];

    TEXT_BUF = .DCB [DCB_A_TEXT_BUF];
    ATTR_BUF = .DCB [DCB_A_ATTR_BUF];
    CHAR_BUF = .DCB [DCB_A_CHAR_SET_BUF];

    CH$FILL (%C' ',
             NUM,
             TEXT_BUF [SRC]);

    CH$FILL (.DCB [DCB_B_DEF_VIDEO_ATTR],
             NUM,
             ATTR_BUF [SRC]);

    IF .CHAR_BUF NEQ 0
    THEN
      CH$FILL (.DCB [DCB_B_DEF_CHAR_SET],
              NUM,
              CHAR_BUF [SRC]);

  END; %;                                     ! end of macro $SMG$BLANK_FILL_DCB
```

!+

\$SMG\$find_nonblank_len

This macro finds the length of a string minus trailing blanks. Since the text is already in a display buffer, there should be no funny characters such as tabs. Start at the end of the string and search backwards to find the first character which is not a space.

```
MACRO $SMG$FIND_NONBLANK_LEN (STRING_ADDR, STRING_LEN, NONBLANK_LEN) =
  BEGIN
    NONBLANK_LEN = STRING_LEN;           ! initialize the length
    WHILE .NONBLANK_LEN NEQU 0
    DO
      BEGIN
        IF (CH$RCHAR (CH$PLUS (STRING_ADDR, .NONBLANK_LEN - 1))
            EQL ' ')
        THEN
          NONBLANK_LEN = .NONBLANK_LEN - 1
        ELSE
          EXITLOOP;
        END;
      END; %;
```

\$SMG\$shuffle

The following macro is used to move text within a display.

```
MACRO $SMG$SHUFFLE (NUM, SRC, DST) =
  BEGIN
    CH$MOVE (NUM,
              TEXT_BUF [SRC],
              TEXT_BUF [DST]);
    CH$MOVE (NUM,
              ATTR_BUF [SRC],
              ATTR_BUF [DST]);
    IF .CHAR_BUF NEQ 0      ! exists only if char set requested
    THEN
      CH$MOVE (NUM,
                CHAR_BUF [SRC],
                CHAR_BUF [DST]);
    END, %;                ! end of macro $SMG$SHUFFLE
```

this is used by SMG\$INSERT_CHARS only

This macro is used to determine how many positions a string will occupy when it is printed. It takes into account funny characters such as tabs and backspaces.

```
MACRO $SMG$FIND_PRINT_LENGTH (TEXT_LEN, TEXT_ADDR, PRINT_LEN) =
  BEGIN
```

```
EXTERNAL LITERAL
  SMGS_FATERRLIB;
```

```
EXTERNAL
  CHAR_TABLE : VECTOR [,BYTE];
```

```
BUILTIN
  SCANC;
```

```
LOCAL
  ALLONES : BYTE INITIAL (-1),
  BYTES_REMAINING, | needed by ref
                   | No. of bytes in input string yet to be
                   | processed.
  IN_POINTER; | Current pointer into input string
```

```
PRINT_LEN = 0; | initialize
BYTES_REMAINING = .TEXT_LEN;
IN_POINTER = TEXT_ADDR;
TEXT_LEN = 0; | modify this to be the actual number
              | of chars processed
```

```
WHILE .BYTES_REMAINING NEQ 0
DO
```

```
  BEGIN ! Overall loop
```

```
  LOCAL
    NEW_BYTES_REMAINING, | No. of bytes remaining as returned
                        | by SCANC
    ADDR_DIFF; | Addr of char in input stream whose
               | index into scanc table yields
               | non-zero code.
```

```
  | +
  | See if any of the remaining input characters require special
  | treatment.
```

```
  SCANC ( BYTES_REMAINING, | No. of bytes remaining
          .IN_POINTER, | Current pointer to source
          CHAR_TABLE, | Address of SCANC table
          ALLONES, | Mask for ANDing
          NEW_BYTES_REMAINING, | New remaining no. of bytes
                               | including the byte which
                               | caused the instruction to
                               | halt. Is zero only if all
                               | bytes did not satisfy search.
          ADDR_DIFF); | Addr of char in input stream
                     | whose index into scanc table
                     | yields non-zero code.
```

```
  IN_POINTER = .IN_POINTER + (.BYTES_REMAINING - .NEW_BYTES_REMAINING);
```

```
  PRINT_LEN = .PRINT_LEN + (.BYTES_REMAINING - .NEW_BYTES_REMAINING);
```

```
  | maintain printing length of chars
```

```
  TEXT_LEN = .TEXT_LEN + (.BYTES_REMAINING - .NEW_BYTES_REMAINING);
```

```
  | pass back no. of chars processed
```

```
  BYTES_REMAINING = .NEW_BYTES_REMAINING;
```

```
  IF .NEW_BYTES_REMAINING EQL 0
  THEN
```


EXITLOOP; ! Break out of loop -- we're done

+ Dispatch on the non-zero code located to see what special
- action is needed.

CASE .CHAR_TABLE [.(.ADDR_DIFF)<0,8>] FROM 1 TO 10 OF
SET

[1,2,3,10]:

Hex Character Codes	ASCII Character
00 to 06	NUL to ACK
0E to 1A	SO to SUB
1C to 1F	FS to US
07	BEL
08	BS
7F	DEL

- Non-printing characters

[4]:

Hex Character Codes	ASCII Character
09	HT

- TABs require extra space.

TAB stops are assumed to be set in the following columns:
9, 17, 25, 33, 41, 49, 57, 65, 73 (width=80)

9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105, 113,
121, 129 (width=132)

BEGIN

PRINT_LEN = .PRINT_LEN +
 ((.DCB [DCB_W_CURSOR_COL]-1)/8+1)*8+1;

TEXT_LEN = .TEXT_LEN + 1; ! include as a 'printable' char

END;

[5,6,7,8,9]:

Hex Character Codes	ASCII Character
0A	LF
0B	VT
0C	FF
0D	CR
1B	ESC

- These characters do not make sense when inserting characters.
Throw away all text after this.

BEGIN

EXITLOOP;

END;

[INRANGE, OUTRANGE]:

```

!+
! Should never get here -- there are no other codes in
! CHAR_TABLE. If we do, we've got a problem.
!-

```

BEGIN

RETURN SMGS_FATERRLIB;

END;

TES;

```

!+
! Re-adjust pointer and count of bytes left to account for
! the special character(s) just processed.
!-

```

IN_POINTER = .IN_POINTER + 1;

BYTES_REMAINING = .BYTES_REMAINING - 1;

END; ! Overall loop

END; %; ! End of macro \$SMGS\$FIND_PRINT_LENGTH

```

!+
! Macros used for input routines.
!-

```

MACRO

\$SMGS\$VALIDATE_KTH (KEY_TABLE_ID, KTH) =

BEGIN

EXTERNAL LITERAL

SMGS_INVKTB_ID;

KTH = .KEY_TABLE_ID [0];

IF .KTH EQL 0

THEN

RETURN SMGS_INVKTB_ID; ! Invalid key-table-id

IF .KTH [KTH_L_CHECK] NEQA .KTH

THEN

RETURN SMGS_INVKTB_ID; ! Invalid key-table-id

END %;

MACRO

\$SMGS\$VALIDATE_KCB (KEYBOARD_ID, KCB) =

BEGIN

EXTERNAL LITERAL

SMGS_INVKBD_ID;

KCB = .KEYBOARD_ID [0];

IF .KCB EQL 0

THEN

RETURN SMGS_INVKBD_ID; ! Invalid keyboard-id

IF .KCB [KCB_L_CHECK] NEQA .KCB

THEN

RETURN SMGS_INVKBD_ID; ! Invalid keyboard-id

END %;

0355

AH-BT13A-SE
VAX/VMS V4.0

**DIGITAL
CONFIDE**

EQUIPMENT
INITIAL AND

CORPORATION
PROPRIETARY

0
Y